# **[mysql的存储过程与事务入门](http://www.cnblogs.com/lisuyun/p/5600155.html)**

存储过程是：  
通过一系列的SQL语句， 根据传入的参数（也可以没有）， 通过简单的调用，   
完成比单个SQL语句更复杂的功能， 存储在数据库服务器端，只需要编译过一次之后再次使用都不需要再进行编译。主要对存储的过程进行控制。  
  
事务是一系列的数据更改操作组成的一个整体。一旦事务中包含的某操作失败或用户中止，用户可以控制将事务体中所有操作撤消，返回事务开始前的状态。  
事务中的操作是一个整体，要么整体完成，要么全部不做。从而保证了数据的完整性。  
Mysql中，MyISAM存储引擎不支持事务，InnoDB支持。

**一、存储过程**

创建一张test1表的存储过程

mysql> delimiter $ -- delimiter $是设置 $为命令终止符号,代替默认的分号,因为分号有其他用处.

mysql> create procedure sp\_test1()

-> begin

-> create table test1(id **int**,name varchar(100));

->insert into test1 values(1,'lilei');

-> select \* from test1;

-> end

-> $

Query OK, 0 rows affected (0.00 sec)

mysql> delimiter ; //恢复分号作为分隔终止符号

调用存储过程

mysql> call sp\_test1();

//MYSQL执行无误

DELIMITER $

DROP PROCEDURE IF EXISTS `sp\_test1`$

CREATE PROCEDURE sp\_test1()

BEGIN

DROP TABLE IF EXISTS test1;

CREATE TABLE test1( id INT,`name` VARCHAR(100));

INSERT INTO test1 VALUES(1,'lilei');

SELECT \* FROM test1;

END$

DELIMITER ;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

创建带参数的存储过程

mysql> delimiter $ -- delimiter $是设置 $为命令终止符号,代替默认的分号,因为分号有其他用处.

mysql> create procedure sp\_test(IN pi\_id **int**, OUT po\_name varchar(10))

-> begin

-> select \* from test.tb\_test;

-> select tb\_test.name into po\_name from test.tb\_test where tb\_test.id = pi\_id;

-> end

-> $

Query OK, 0 rows affected (0.00 sec)

mysql> delimiter ; -- 恢复分号作为分隔终止符号

5.调用存储过程

mysql> set @po\_name='';

Query OK, 0 rows affected (0.00 sec)

mysql> call sp\_test(1,@po\_name);

Select @po\_name; -- 对变量值进行查看

## 二、事务处理

mysql中如果是不支持事务的引擎，如myisam，则是否commit都没有效的。  
如果是支持事务的引擎，如innodb，则有系统参数设置是否自动commit，查看参数如下：  
mysql> show variables like '%autocommit%';  
+---------------+-------+  
| Variable\_name | Value |  
+---------------+-------+  
| autocommit | ON |  
+---------------+-------+  
1 row in set (0.00 sec)  
  
mysql>  
  
显示结果为on表示事务自动提交，也即不用手工去commit。  
当然，你可以设置其为OFF，然后自己手工去commit。  
  
使用了commit后你不可以回退，它会永久改变你的数据，而不使用只是临时改变，它把改变后的内容备份了一份在日志中，你可以rollback还原回来，看情况而用了。

    事务中可以有存储过程 存储过程中也可以有事务

 事务是一系列的对数据库的操作，这些操作包括存储过程，更改语句及其它操作。开始语句是start TRANSACTION (事务开始），结束语句有两种，一个是ROLLBACK--回滚，一个是--commit提交事务的所有操作。

存储过程是实现一定功能的语句组成的程序段。可以包括事务，也可以在某事务当中。

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

对于MySQL数据库，如果你要使用事务以及行级锁就必须使用INNODB引擎。如果你要使用全文索引，那必须使用myisam。 INNODB的实用性，安全性，稳定性更高但是效率比MYISAM稍差，但是有的功能是MYISAM没有的。修改MySQL的引擎为INNODB，可以使用外键，事务等功能，性能高。

查看数据表的状态:

SHOW TABLE STATUS FROM sites WHERE NAME='site';    
SHOW TABLE STATUS FROM db\_name WHERE NAME='table\_name';

更改表的引擎: innodb----myisam

alter table table\_name engine=innodb;    
alter table table\_name engine=myisam;

Start transaction;

SELECT \* FROM person;

INSERT INTO person VALUES('a',11,12);

INSERT INTO person VALUES('b',12,13);

-- COMMIT; --进行测试 提交或者回滚

-- ROLLBACK;

代码中可通过判断mysql语句执行成功否 进行相应的回滚操作。

If（DB.Query（“insert into table\_name values(‘’,’’)”）==false）

DB.execute(“Rollback”);

## 函数

简单函数：

DELIMITER $$

CREATE FUNCTION hello\_world()

RETURNS TEXT

LANGUAGE SQL

BEGIN

RETURN 'Hello World';

END;

$$

DELIMITER ;

带参数函数：

DROP FUNCTION IF EXISTS hello\_world;

DELIMITER $$

CREATE FUNCTION hello\_world(addressee TEXT)

RETURNS TEXT

BEGIN

DECLARE strlen INT;

SET strlen = LENGTH(addressee);

RETURN CONCAT('Hello ', addressee, ' - your parameter has ', strlen, ' characters');

END;

$$

DELIMITER ;

SELECT hello\_world('fd')

-----Reult：

Hello fd - your parameter has 2 characters

循环函数：

DROP FUNCTION IF EXISTS looptest;

DELIMITER $$

CREATE FUNCTION looptest()

RETURNS INT READS SQL DATA

BEGIN

DECLARE v\_total INT;

SET v\_total = 0;

count\_loop: LOOP

SET v\_total = v\_total + 1;

IF v\_total = 10 THEN

LEAVE count\_loop;

END IF;

END LOOP;

RETURN v\_total;

END;

$$

DELIMITER ;

# PG 相关内容

## 存储过程与函数：

<https://blog.csdn.net/rachel_luo/article/details/8073458>

**CREATE** **OR** **REPLACE** **FUNCTION** public."UpdateValueByThre"(thre **numeric**(18,6),tablename **varchar**(32),fd **varchar**(32))

**RETURNS** void

LANGUAGE plpgsql

**AS** **$$**

**declare** i **int**:=1;

vf **varchar**(32);

**BEGIN**

while i < 1008 LOOP

vf = **concat**('update ',tablename,' set ',fd,i,'=if(',fd,i,'<',thre,',1,0);');

**execute** vf;

i = i + 1;

**end** LOOP;

**END**

**$$**;

Querydate\_sql = ‘select a from b’;

**execute** querydate\_sql **into** t6; //将某个值传入变量

PG中文在线文档 [http://postgres.cn/docs/10/xfunc-sql.html#XFUNC-OUTPUT-PARAMETERS](http://postgres.cn/docs/10/xfunc-sql.html" \l "XFUNC-OUTPUT-PARAMETERS)

Example:

**1.返回sql结果集**

**create** **or** **replace** **FUNCTION** sp\_showgis\_singlenetwork(tablename **varchar**(64),showsql **varchar**(1080))

**RETURNS** **table**(enodebid **int**,cid **int**,kpi **int**) **AS** **$$**

**declare**

checksql **varchar**(256);

countNum **int**:=0;

exeSQL **varchar**(1080);

**BEGIN**

checksql := 'SELECT count(\*) FROM ' || tablename;

**execute** checksql **into** countNum;

**if** countNum >0 **then** exeSQL=showsql;

**else** exeSQL='select 1,1,1'; **end** **if**;

**return** query **execute** exeSQL;

**END**;

**$$** **LANGUAGE** plpgsql;

运行：

**select** \* **from** sp\_showgis\_singlenetwork('nr\_projpara','select 1,2,3 from nr\_projpara where \_primary=''460''')

总结：

Return table（列名以及数据类型必须与返回SQL一致）

可改为 SETOF foo

CREATE FUNCTION getfoo(int) RETURNS SETOF foo AS $$

SELECT \* FROM foo WHERE fooid = $1;

$$ LANGUAGE SQL;

SELECT \* FROM getfoo(1) AS t1;

那么我们会得到：

fooid | foosubid | fooname

-------+----------+---------

1 | 1 | Joe

1 | 2 | Ed

(2 rows)

**2.返回单行结果集record**

**CREATE FUNCTION getfoo(int) RETURNS foo AS $$**

**SELECT \* FROM foo WHERE fooid = $1;**

**$$ LANGUAGE SQL;**

**SELECT \*, upper(fooname) FROM getfoo(1) AS t1;**

**3.默认参数，返回int**

**CREATE** **FUNCTION** foo(a **int**, b **int** **DEFAULT** 2, c **int** **DEFAULT** 3)

**RETURNS** **int**

**LANGUAGE** **SQL**

**AS** **$$**

**SELECT** **$1** + **$2** + **$3**;

**$$**;

运行：

**select** foo(1)